CS2010 PS1 - Baby Names v4 (with R-option)

Released: Friday, 22 August 2014, 8am
Due: Saturday, 06 September 2014, 8am

Collaboration Policy. You are encouraged to work with other students or teaching staffs (inside or outside this module) on solving this problem set. However, you **must** write Java code **by yourself**. In addition, when you write your Java code, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). This list may include certain posts in CS2010 Facebook group. If you have access to your seniors' CS2010 files (that is, problem sets version 1/2/3), please refrain from looking at their code verbatim. Any deviation from this policy will be considered as cheating. If the offender is caught beyond reasonable doubt, he/she will be punished severely, including referral to the NUS Board of Discipline. It is not worth it to cheat just to get 19% when you will lose out in the other 81%.

R-option. This PS has R-option at the back. This additional task usually requires understanding of data structure or algorithm *beyond* CS2010. A self research on those relevant additional topics will be needed but some pointers will be given. CS2010R students *have to* attempt this R-option. CS2010 students can choose to attempt this R-option too for extra challenge, or simply leave it.

Background. This is the first problem set for CS2010/R. As you may have known from the first lecture, your lecturer Steven is already a father since three years ago (to be precise, since the arrival of his first baby on 24 October 2011) and will remain a father for the rest of his life (having his second baby on 16 July 2014 does NOT change this fatherhood status). Three years ago, Steven has decided to give a grand theme to his CS2010: 'Babies'. Many, if not all of you, will probably encounter these series of 'real life problems' if you have wife/husband and bab(ies) in a few years time. Since this 'value added feature' of CS2010 is generally well received among your seniors in the past three years, Steven has decided to stick with the same storyline, although now they are all past events. Steven hopes that CS2010 students find this semester a bit more special by knowing that all these 'real life problems' can be 'solved' with what you learn in CS2010. Note: PS1 to PS7 are chronological. PSBonus is not part of this storyline.

2011 Story (repeated in 2014). PS1 happened when Steven and his wife (Grace Suryani) discovered that Grace was pregnant (this was back in mid February 2011). After confirming the result of our home pregnancy test kit (the 'two stripes' == pregnant) with a gynaecologist, we started to break this news to our parents, to my brother, then to our friends. Soon, we encountered this situation: My parents gave a few baby name suggestions, my wife's parents did the same thing, and so did some of our friends. We ourselves had a few ideas for our baby's name.

In the past (when Steven was a baby himself), nobody knows the gender of a baby until he/she is born. Today's medical technology (ultrasound scan) can detect the gender of the baby roughly around the second trimester of pregnancy. Thus, at the early stage of pregnancy when we still did not know the gender of our baby, we received suggestions for both male and female baby names.

Eventually, we had to select one name. Steven, being a computer scientist, wanted to use his Computer Science knowledge to help him answering some queries.

Figure 1: One of the first photos of Jane (2011) and Joshua (2014)

Steven and his wife, after knowing that the gender of our first baby is female (at around June 2011), eventually decided that her name will be *"Jane Angelina Halim"*. "Jane" means "God is Gracious", "Angelina" means "God's messenger", and "Halim" is Steven's family name.

Then, in 2014, after knowing that the gender of our second baby is male and because we want our children have similar name abbreviation: 'JAH' and 'JBH', eventually decided that his name will be *"Joshua Ben Halim"*. "Joshua" means "God is my salvation", "Ben" means "Son (a male children)", and the standard "Halim" – Steven's family name.

The Actual Problem. Given N *distinct* baby name suggestions (each baby name consists of *only* uppercase alphabet characters of no more than $\mathbf{M = 30}$ **characters**) and the gender suitability of that name (integer 1 for male or integer 2 for female), tell Steven how many baby names start with a *prefix*[1] that is inside a given query interval `[START..END)`, where[2] `START` < `END`, and both are strings. Notice that the interval is left-closed and right-open. There are Q queries that you have to answer. Use the most efficient technique that you have learned so far.

The skeleton program `BabyNames.java` is already written for you. You just need to implement mainly these three more methods/functions:

- `void AddSuggestion(String babyName, int genderSuitability)`
  Insert `babyName` and its `genderSuitability` into a data structure of your choice.
- `int Query(String START, String END, int genderPreference)`
  Query your data structure and report the number of baby names that start with a prefix that is inside the query interval `[START..END)`, depending on parameter `genderPreference`:
  - If `genderPreference = 0`, report the number of both male and female baby names.
  - If `genderPreference = 1`, report the number of male baby names only.
  - If `genderPreference = 2`, report the number of female baby names only.

  Note that `AddSuggestion` and `Query` operations **can be interleaved**.
- If needed, update the constructor of Class `BabyNames`.

[1] A prefix of a string $T = T_0T_1...T_{n-1}$ is string $P = T_0T_1...T_{m-1}$ where $m \leq n$.
[2] In Java, you can compare two strings using `compareTo` method.

Examples:
Let there be $N = 4$ distinct baby names: {(JANE, 2), (JOSHUA, 1), (MARIA, 2), (PETER, 1)}.

- `Query(''PET'', ''STE'', 1)` = 1 as we have (PETER, 1).
- `Query(''PET'', ''STE'', 2)` = 0 because although we have PETER, it is *not* a female baby name.
- `Query(''JA'', ''PETI'', 0)` = 4 as we have (JANE, 2), (JOSHUA, 1), (MARIA, 2), and (PETER, 1).
- `Query(''JA'', ''PETA'', 0)` = 3 as we have (JANE, 2), (JOSHUA, 1), (MARIA, 2). Notice that PETER is *outside* the query interval `['JA'..'PETA')` as 'PETER' $\geq$ 'PETA'.
- `Query(''JOSH'', ''PET'', 1)` = 1 as we have (JOSHUA, 1). Notice that PETER is outside the query interval `['JOSH'..'PET')` as 'PETER' $\geq$ 'PET'. Remember that the interval is left-closed and right-open.
- `Query(''JANE'', ''MARIA'', 2)` = 1 because JANE is a female baby name that has prefix inside the query interval `['JANE'..'MARIA')`, but MARIA is not as 'MARIA' $\geq$ 'MARIA' (actually they are equal). Remember that the interval is right-open!
- `Query(''JANE'', ''MARIANA'', 2)` = 2, now JANE and MARIA are inside the query interval `['JANE'..'MARIANA')` as 'MARIA' $<$ 'MARIANA'.

Now, if we add one more baby name {(CHLOE, 2)} so that we have $N = 5$ and we ask `Query(''A'', ''ZZZ'', 0)`, we have an answer 5.

Subtask A (25 points). $Q \leq 10$, $N \leq 26$. All baby names have distinct first letter. Both `START` and `END` only contains 1 character (the maximum `END` is thus 'Z').

Subtask B (Additional 50 points). $Q \leq 20000$, $N \leq 20000$. With $N \leq 20000$ (yes, there are lots of different name suggestions for Steven and Grace), there will obviously be several names with the same first letter. Now, `START` and `END` can have more than one characters. The gap between `START` and `END` is a mix between small range (e.g. `[''SA''..''STR'')`, `[''PE''..''PO'')`, etc) and large range (e.g. `[''A''..''ZZ'')`, `[''AB''..''YZ'')`, etc). Subtask B has a very loose time limit setting of 12 seconds (the official solution runs in under 1 second) so that students can focus on correctness first before dealing with runtime speed.

Subtask C (Additional 18 points). To get up to 93 points in this PS, you have to solve Subtask B above with *your own* balanced Binary Search Tree (that is, your solution code cannot use Java TreeMap or TreeSet at all!). The official test data and the time limit setting is the same as Subtask B above but your Lab TA will verify if your *last* submission to Subtask C indeed use the proper balanced Binary Search Tree written by yourself and award 18 marks manually. Solutions that do not use balanced Binary Search Tree will be given 0 point for this Subtask C.

Subtask D (Additional 7 points). To get up to 100 points in this PS, you have to fulfill the requirement of Subtask B and C above, **but with a very strict time limit of 1s**. That is, you need to match our current best official solution that runs very fast ($\approx 0.2s$) on the same test data. We are aware that this Subtask D can be very frustrating with lots of 'Time Limit Exceeded' verdicts. Please balance your urge to finish this Subtask D with the assignments from your other modules. There is a way to recover these 7 points via Subtask A of PS Bonus during recess week later.

R-option/Subtask E (No bonus point for non CS2010R students). Given N *distinct* baby name suggestions (each baby name consists of *only* uppercase alphabet characters of no more than M characters), tell Steven how many baby names **contain *substring*[3] that match query substring** `SUBSTR`. There are Q queries that you have to answer **after you are given all N distinct baby names**. Note that for this R-option, we have $Q \leq 10000$, $N \leq 10000$, but this time a baby name can go up to $M = 100$ characters. However, we guarantee that the answer for each query will not be larger than 100.

The skeleton program `BabyNamesR.java` is given. The important modifications are:

- `void AddSuggestion(String babyName)`
  We drop `genderSuitability`.
- `int Query(String SUBSTR)`
  We change `START` and `END` to just one parameter: `SUBSTR` and drop `genderPreference`.
  We guarantee that this function will never report answer larger than 100.

Using these 5 *distinct* baby names: ETERNIA, JANE, JOSHUA, MARIA, and PETER(ignoring their gender suitability), we give some examples of the modified queries (this time the query only has one parameter) below:

- `Query(''ER'')` = 2 as we have ET<u>ER</u>NIA and PET<u>ER</u>.
- `Query(''E'')` = 3 as we have <u>E</u>T<u>E</u>RNIA, JAN<u>E</u>, and P<u>E</u>T<u>E</u>R. Note that there are two character 'E' in 'PETER' and 'ETERNIA', but we only count these two names once.
- `Query(''IA'')` = 2 as we have ETERN<u>IA</u> and MAR<u>IA</u>.
- `Query(''JOSHUA'')` = 1 as we only have one <u>JOSHUA</u>.
- `Query(''ETER'')` = 2 as we have <u>ETER</u>NIA and P<u>ETER</u>.

Note that your solution for the original problem will be wrong for this R-option requirement. Therefore, a student who want to attempt this R-option has to submit a different `BabyNamesR.java`. Students who take both CS2010 and CS2010R have to attempt both versions (this can be very tedious so you have to plan your time carefully, maybe by skipping Subtask D).

Note 1: If you encounter stack overflow, try running your Java program with: '-Xss8m' flag. Ask your Lab TA if you are not sure about the meaning of this flag.

Note 2: The official test data has been uploaded to Mooshak online judge, but it is hidden from your view. The time limit setting in Mooshak online judge for Subtask A, B, C, D, and E (R-option) are 1, 12, 12, 1, and 5 seconds, respectively. You are encouraged to generate and post additional test data in Facebook group. Please use `BabyNamesVerifier.java` to verify whether your custom-made test data (for Subtask A+B+C+D) conform with the required specifications.

Note 3: You will likely submit multiple versions of Java code for this PS (as there is no submission limit for our PSes). However, your lab TA will *only* grade your **LAST** submission. Therefore, please make sure that your last submission is the one that you want to be graded. For this PS, if your last submission to Subtask C+D (accidentally) uses Java TreeMap or TreeSet, your submission for Subtask C+D will get a straight 0 point.

[3]A substring of a string $T = T_0T_1...T_{n-1}$ is string $P = T_iT_{i+1}...T_j$ where $0 \leq i \leq j < n$.